Q-Learning and Deep Q-Learning Approaches to the Airport Gate Assignment Problem

Hakyong Kim*

Abstract

This project presents Markov decision process formulation for the airport gate assignment problem and based on this formulation, we applied Q-Learning and deep Q-Learning. In Q-Learning, the categorization function was utilized to reduce the size the state space. In deep Q-Learning, on the other hand, double DQN with fully connected, 2 hidden-layered network was utilized. Methods were tested on two types of problem instances. And we compare the results with greedy algorithm and Branch and Price algorithm. Q-Learning was turned out to be an inadequate approach for the problem and the policy derived from deep Q-Learning was greedy policy.

1 Introduction

The airport gate assignment problem (AGAP) is the problem of assigning flights to compatible gates in airport. When flights arrive at airport, each flight needs to be assigned to a gate to prepare for the next scheduled flight. Since gate assignment is coupled with relevant airport work schedule, it is not just a matter of passengers' utility but also of airport operational schedule. As the interconnection between worldwide countries keeps growing, air traffic increased greatly in recent decades and the way how flights are assigned to gates becomes more crucial for both passengers' comfort and airport/airlines' operational efficiency [1].

Therefore, AGAP is an important issue in airport management system. To deal with this issue, various types of AGAP have been studied. In terms of formulation, objective functions of minimizing the total walking distance, discomfort, waiting time of passengers (passenger oriented) and minimizing the number of un-gated flights, total arrival delays, operational cost (airport/airlines oriented) etc, were mainly considered [1]. In terms of methods, most of works were based on heuristic, metaheuristic, optimization approaches and, as far as we know, only two study [2,3] have been conducted for AGAP which adopted the reinforcement learning as a methodology. [2] considered AGAP whose objective function is minimizing the total walking distance and applied a probability learning based tabu search. Probability learning scheme is learning automata (LA) which is a policy iteration method. [3] considered AGAP that minimizes the number of un-gated flights and used policy gradient method via deep neural network.

Recently, [4] suggest a compact mixed-integer-programming (MIP) model for AGAP that considers minimizing the total arrival delays. By extending the compact MIP model, [4] made a set covering master problem to utilize a column-generation technique together with submodular maximization, dynamic programming, approximate dynamic programming, etc. Experiment results show that this approach quickly finds an optimal solution for small-sized instances but it takes several hours for medium and large-sized instances when optimality gap was set to 2%.

Optimization based method has advantages in that it gives the dual objective function value, that is, it can provide the quality of the solution it has found. And it searches for the optimal solution although it may be intractable. However optimal solution of one problem instance cannot be generalized to another problem

*Department of Industrial Engineering, Seoul National University(gkrdyd111@snu.ac.kr).

instance and thus needs to be solved again from scratch. On the other hand, output of reinforcement learning is a policy and not a direct solution. So the policy derived from reinforcement learning can be applied to many problem instances if the distribution of their parameters are not that different from those of training data. Therefore it is worth-while to apply reinforcement learning to AGAP that minimizes the total arrival delays (problem of [4]) which has never been done before. Motivated by the advantage of reinforcement learning it has over optimization approaches and its recent application to combinatorial optimization problems [5] , we applied Q-Learning and double deep Q-Learning (DQN) for AGAP.

1.1 Contribution

Our project has the following three major contributions.

1. We formulated a MDP model for AGAP and applied Q-Learning, double DQN respectively. This is the first Q-Learning and DQN approaches for AGAP.
2. In Q-Learning, we proposed a novel state reduction method, to overcome the curse of dimensionality.
3. By comparing the results of double DQN with greedy algorithm and Branch and Price algorithm, we could interpret the behavior of the policy derived from double DQN.

2 Problem Description

2.1 Optimization model

AGAP considered in this project is the one that minimizes the total arrival delays. In this project, we refer to AGAP as the one considered in this project, not the general one. Compatibility between gates and flights is not considered since each compatible flights and gates can be decomposed into separable independent problems. Compact MIP formulation for this AGAP is a simplified version of the one in [4]. Figure 1 illustrates one example of AGAP. In figure 1, only one arrival delay($t_4 - h_4$) is occurred until flight 7. A formal description of MIP formulation is as below.

Notation

- Sets
  F : Set of flights (sorted in chronological order of $h_i's$).
  G : Set of gates.
- Decision variables
  x_{ik} : Gate assignment variable. 1 if flight i is assigned to gate k.
  t_i : Park time of flight i at its assigned gate.
- Parameters
  p_i : Processing time of flight i, which includes buffer time and turn time. Integer.
  h_i : Arrival time of flight i. Integer.
  M : Arbitrary big number. Big-M.

MIP formulation

$$\text{minimize} \quad \sum_{i \in F} (t_i - h_i) \tag{1}$$

$$\text{subject to} \quad \sum_{k \in G} x_{ik} = 1, \quad \forall i \in F, \tag{2}$$

$$t_i \geq h_i, \quad \forall i \in F, \tag{3}$$

$$t_i + p_i - t_j \leq M(2 - x_{ik} - x_{jk}), \quad \forall i < j, \ \forall i, j \in F, \ \forall k \in G, \tag{4}$$

$$x_{ik} \in \{0, 1\}, \quad \forall i \in F, k \in G, \tag{5}$$

$$t_i \geq 0, \quad \forall i \in F. \tag{6}$$

Objective function (1) is the sum of all arrival delays. Constraint (2) ensures that flight i must be assigned to exactly one gate. Constraint (3) ensures that flight i can be parked to its assigned gate after its arrival time. Constraint (4) ensures that subsequent flight j of flight i in the same gate can be parked after the processing time of flight i. Constraints (5) and (6) ensures that decision variables x and t must be binary and non-negative respectively.

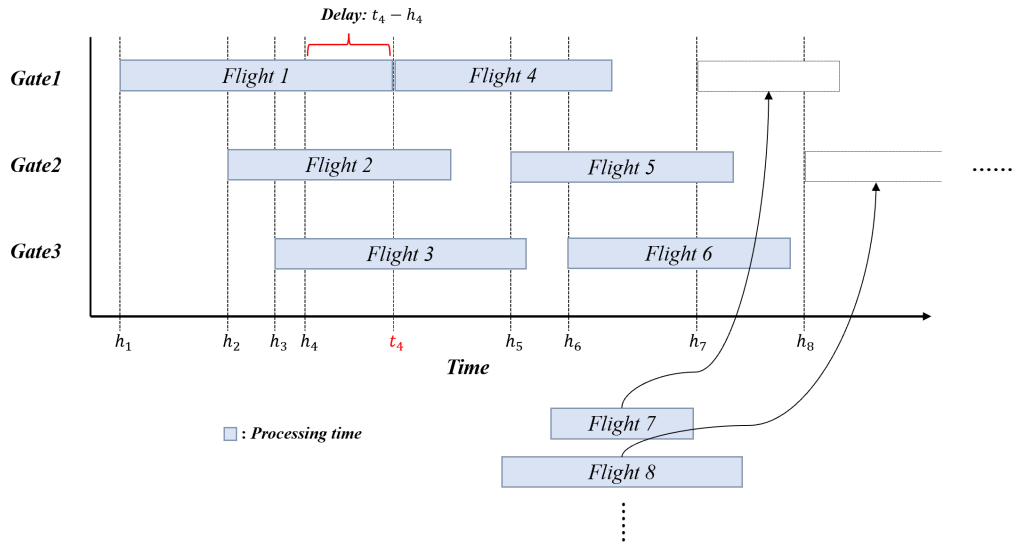

Figure 1: Illustration of AGAP

2.2 MDP for AGAP

AGAP can be formulated as finite Markov decision process (MDP) which is sequential decision making process. In MDP, actions are flight-to-gate assignments which correspond to decision variables x in MIP model. Each action a_t is determined sequentially in order of arrival time of flight, h_i. Since we have total $|F|$ flights, we have to determine $|F|$ actions sequentially, and thus the number of stages is $|F|$. State is defined as $s_t = (g_t^1, \cdots, g_t^{|G|})$ where g_t^k is the delay time we may have at stage t if flight t is assigned to gate k, i.e. $a_t = k$. This definition of state gives information about the amount of congestion of each gate at the current stage. Since the objective of this problem is to minimize the total arrival delay, reward is defined as the negative of the arrival delay we earn by taking action at the current stage. Figure 2 illustrates the environment of figure 1 at stage 7 in the form of suggested MDP. If flight 7 is assigned to gate 1, no arrival delay is incurred and we get 0 reward. And then time is jumped h_8 of stage 8.

MDP formulation

- **State**
  $S = \mathrm{Z}_+^{|G|}$
  $s_t = \{(g_t^1, \cdots, g_t^{|G|}) : g_t^k \in \mathrm{Z}_+,\ \forall k \in G\}$
- **Action**
  $A = \{1, \cdots, |G|\}$
  $a_t = k$ represents assigning flight t to gate k.
- **Reward**
  $r(s_t, a_t) = -g_t^{a_t}$
- **System Equation**
  $s_{t+1} = (g_{t+1}^1, \cdots, g_{t+1}^{|G|})$ where $g_{t+1}^k = \begin{cases} \max\{g_t^k - (h_{t+1} - h_t), 0\}), & k \neq a_t, \\ \max\{g_t^k + p_t - (h_{t+1} - h_t), 0\}), & k = a_t, \end{cases}$.

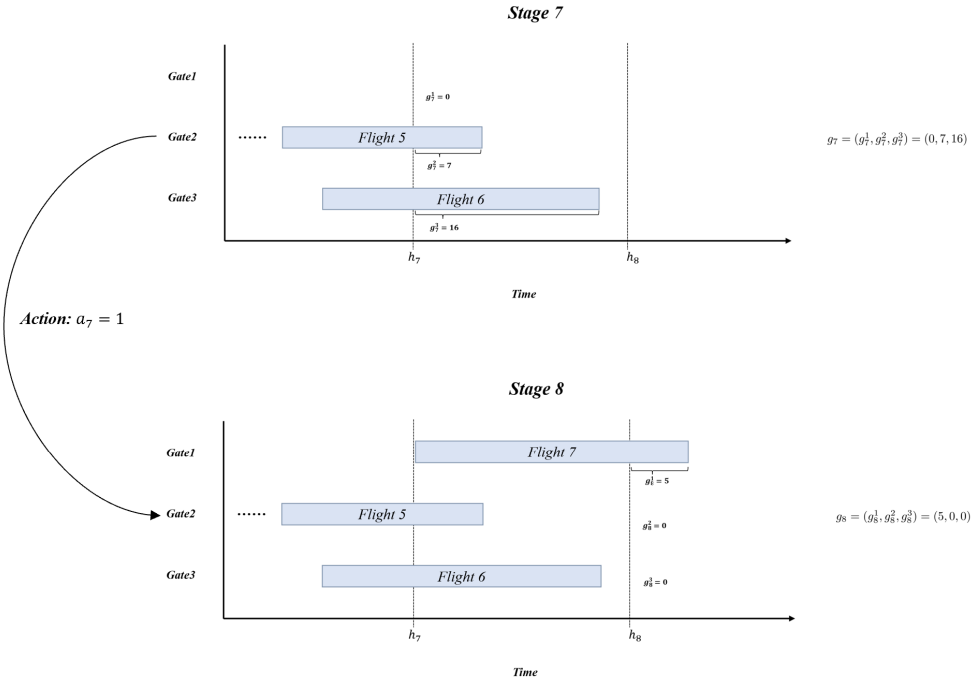

Figure 2: Illustration of MDP

3 Method

3.1 Q-Learning

To apply the Q-Learning to AGAP, it is necessary to reduce the size of state space otherwise state space will blow up due to curse of dimensionality. So we categorize the amount of arrival delay into a few numbers so that we can make adequate size of state space for Q-Learning. We define $d : \mathrm{Z}_+ \to \{0, 1, \cdots, N-1\}$ by a categorization function that converts actual arrival delay to category of arrival delay where N is the number of category. For example, arrival delay was categorized as 0 if the actual value was 0, 1 if the actual value was between 1 to 5, and 2 if the actual value was over 5. In this case, the size of state space becomes $3^{|G|}$ which is affordable size when $|G|$ is small.

If the categorization function is used, the actual arrival delay cannot be expressed by action or state. At stage t, the history of actions before stage t is stored in environment. Based on this information, the schedule before stage t can be easily constructed and thus, we can calculate the earliest possible time to assign flight t to gate k, W_t^k. W_t^k is used for calculating reward and updating state.

MDP formulation for Q-Learning

- **State**
  $S = \{0, \cdots, N-1\}^{|G|}$
  $s_t = \{(g_t^1, \cdots, g_t^{|G|}) : g_t^k \in \{0, \cdots, N-1\},\ \forall k \in G\}$
- **Action**
  $A = \{1, \cdots, |G|\}$
  $a_t = k$ represents assigning flight t to gate k.
- **Reward**
  $r(s_t, a_t) = -\max\{W_t^{a_t} - h_t, 0\}$
- **System Equation**
  $s_{t+1} = (g_{t+1}^1, \cdots, g_{t+1}^{|G|})$ where $g_{t+1}^k = \begin{cases} d(\max\{W_t^k - h_{t+1}, 0\}), & k \neq a_t, \\ d(\max\{W_t^k + p_t - (h_{t+1} - h_t), 0\}), & k = a_t. \end{cases}$
  $w_{t+1} = (w_{t+1}^1, \cdots, w_{t+1}^{|G|})$ where $w_{t+1}^k = w_t^k + \mathbb{1}[a_t = k]\, p_t, \quad \mathbb{1}$: indicator function.

Q-learning can be easily applied to the above MDP formulation. Stochastic approximation was used for updating Q-value. Visitation count was used to compute stepsize α. We adopted ϵ-greedy policy to handle the exploration issue.

Algorithm 1 Q-Learning

Initialize $Q(s,a) := 0,\ N(s,a) := 0, \quad \forall s \in S,\ a \in A$
for $t = 1$ to T **do**
  Initialize S
  repeat
    Choose A by ϵ-greedy policy
    Take action A, observe R, S'
    $N(S,A) \leftarrow N(S,A) + 1$
    $Q(S,A) \leftarrow Q(S,A) + \frac{1}{N(S,A)^p}[R + \gamma \max_{A'} Q(S', A') - Q(S,A)]$
    $S \leftarrow S'$
  until S is terminal
end for

3.2 Deep Q-Learning

Based on MDP formulation for AGAP, we can easily utilize DQN. In DQN approach, we don't have to categorize the amount of arrival delay, g_t^k, since we can exploit the expressiveness of deep neural network. For better stability in training, double DQN was used. In this project, we tested two different MDP formulation for double DQN. One is the MDP formulation in section 2.2 (basic MDP) and the other is the MDP formulation with additional state features (feature-added MDP).

Additional features can be added to state to capture various information in data such as h_t, p_t. [6] suggests 7 features for state representation in reinforcement learning environment for job-shop scheduling. These features can be applied to our problem since AGAP is a special problem of scheduling problem and thus job-shop scheduling problem in [6] and AGAP are closely related. For example, $v_t^k = \frac{g_t^k + p_t}{h_{t+1} - h_t}$, $\frac{g_t^k + p_t}{h_{t+2} - h_t}$ will

be helpful for making decisions in subsequent stages. Also, statistic vector (u_t) such as mean and variance of parameters of leftover flights can be added.

MDP formulation for deep Q-Learning

- **State**
  $S = \mathrm{R}^{|G|} \times \mathrm{R}^{dim(v)} \times \mathrm{R}^{dim(u)}$
  $s_t = \{(g_t, v_t, u_t) : g_t \in \mathrm{R}^{|G|},\ v_t \in \mathrm{R}^{dim(v)},\ u_t \in \mathrm{R}^{dim(u)}\}$ where
  $g_t = (g_t^1, \cdots, g_t^{|G|})$: arrival delay vector at stage t,
  $v_t = (v_t^1, \cdots, v_t^{|G|})$: additional feature vector at stage t (Gate specific feature),
  u_t : statistic vector of leftover flights at stage t (Gate independent feature).

- **Action**
  $A = \{1, \cdots, |G|\}$
  $a_t = k$ means assigning flight t to gate k.

- **Reward**
  $r(s_t, a_t) = -g_t^{a_t}$

- **System Equation**
  $s_{t+1} = (g_{t+1}, v_{t+1}, u_{t+1})$ where $g_{t+1}^k = \begin{cases} \max\{g_t^k - (h_{t+1} - h_t), 0\}), & k \neq a_t, \\ \max\{g_t^k + p_t - (h_{t+1} - h_t), 0\}), & k = a_t, \end{cases}$
  and v_{t+1}, u_{t+1} depend on how v, u are defined.

Algorithm 2 Double DQN

Initialize network parameters ϕ and ϕ^-
repeat
  Collect dataset $\{(S_i, A_i, S_i', R_i)\}$ using ϵ-greedy policy and add them to **Buffer**
  for $k = 1$ to K **do**
    Sample a mini-batch $\{(S_j, A_j, S_j', R_j)\}$ from **Buffer**
    $y_j^- := r_j + \gamma Q_{\phi^-}(S_j', \underset{A'}{\operatorname{argmax}}\, Q_\phi(S_j', A')), \quad \forall j$
    $\phi \leftarrow \phi - \alpha \sum_j \frac{dQ_\phi}{d\phi}(Q_\phi(S_j, A_j) - y_j^-)$
    $\phi^- \leftarrow (1 - \tau)\phi^- + \tau\phi$
  end for
until ϕ converge

4 Experiment

We conducted experiments to investigate the performance of Q-Learning and double DQN. In addition, greedy algorithm and Branch and Price algorithm were conducted to compare the performance and better understand the behavior of our proposed methods. Next, we consider four test instances, which have same problem size but different distribution of parameters.

4.1 Experiment Details

Greedy algorithm chooses action (a_t, i.e. gate) that has the smallest value of g_t^k at each stage t. Branch and Price algorithm is based on [4] with a slight modification in solving pricing problems. All methods except Branch and Price algorithm were implemented in Python 3.9 and Branch and Price algorithm was implemented in Xpress 8.9. Although double DQN was conducted on two types of MDP formulation (basic MDP, feature-added MDP), only the results of basic MDP were reported because feature-added MDP showed no advantage over basic MDP and sometimes showed worse performance compared to basic MDP. Explanation for this behavior will be discussed in later section.

Test instance. All four test instances have $|F| = 30$, $|G| = 5$. Four test instances are divided into two groups, Type I and Type II. Type I instances have equal inter-arrival time ($h_{t+1} - h_t$) and processing time (p_t) and as a result, Type I instances can be optimally solved by greedy algorithm. Type II instances have inter-arrival time and processing time of uniform distribution and thus, Type II instances cannot be optimally solved by greedy algorithm.

Table 1: Test instance parameter.

	Type I		Type II	
	Instance 1	Instance 2	Instance 3	Instance 4
Inter-arrival time	2	2	U(0,4)	U(0,4)
Processing time	9	15	U(4,14)	U(10,20)

Note. U represents integral discrete uniform distribution.

Q-Learning. Parameters for Q-Learning were set to $\epsilon = 0.5,\ \gamma = 1,\ p = 0.50001$. For categorization functions, five functions ($d1$, $d2$, $d3$, $d4$, $d5$) were tested.

Table 2: Categorization functions.

Interval	$N = 3$		$N = 4$		$N = 5$
	$d1$	$d2$	$d3$	$d4$	$d5$
0	0	$[0, 5]$	0	$[0, 10]$	$[0, 7]$
1	$(0, 5]$	$(5, 10]$	$(0, 10]$	$(10, 20]$	$(7, 14]$
2	$(5, \infty)$	$(10, \infty)$	$(10, 20]$	$(20, 40]$	$(14, 21]$
3	-	-	$(20, \infty)$	$(40, \infty)$	$(21, 28]$
4	-	-	-	-	$(28, \infty)$

Double DQN. Network structure for ϕ, ϕ^- consists of *input layer* ($|S|$) - *hidden layer1* (256) - *hidden layer2* (256) - *output layer* ($|A|$). We train our models with Adam optimizer. We use learning rate of 10^{-6}. discount factor $\gamma = 0.99$, mini-batches of 128 sequences. For computational efficiency and stability, experience replay and target Q-network was utilized. Q-network was updated by means of Polyak averaging with rate of 10^{-6}. Initial exploration rate ϵ was set to 1 and linearly decayed to 0.02.

4.2 Results and Analysis

Q-Learning. Q-Learning could learn optimal policy only for instance 1. Since more categorization number, i.e. N, allows larger search spaces, as N increases, total arrival delay tends to decrease for all instances. Even for the same categorization, the way how intervals are set greatly influenced the result. However, due to the loss of information in search space categorization, Q-Learning showed poor performance generally.

Table 3: Total arrival delay of Q-learning.

Instance Type	Instance	$d1$	$d2$	$d3$	$d4$	$d5$
I	1	0	18	0	0	60
	2	5655	5655	431	387	384
II	3	95	85	81	132	76
	4	5929	1086	777	534	500

Double DQN. Double DQN could learn optimal policy for instance type I, but couldn't for type II. Except for instance 3, total arrival delays of double DQN were same as greedy algorithm. And even for instance 3, it gave the same total arrival delay of greedy algorithm after a parameter tuning. This result implies that the policy derived from double DQN is just at best the greedy algorithm. This could be verified by inspecting the flight-to-gate assignment of double DQN and greedy algorithm. Thus, policy derived from double DQN was far from optimal policy.

Table 4: Total arrival delay of three different approaches.

Instance Type	Instance	*Double DQN*	*Greedy algorithm*	*Branch and Price algorithm*
I	1	0	0	0
	2	375	375	375
II	3	65	56	49
	4	484	484	377

Note. Value of Branch and Price algorithm is the optimal value.

Table 5: Flight-to-gate assignments of instance 4.

	Double DQN	*Greedy algorithm*	*Branch and Price algorithm*
Gate 1	[3,6,11,18,23,28], 99	[0,5,12,15,20,27], 98	[1,6,17,18,21,28], 84
Gate 2	[0,5,12,15,20,27], 98	[1,6,11,18,23,28], 99	[3,7,12,16,19,20,27], 83
Gate 3	[4,9,14,19,24,29], 108	[2,8,13,17,21,26], 91	[4,11,15,22,25,26], 58
Gate 4	[1,7,10,16,22,25], 88	[3,7,10,16,22,25], 88	[5,14,23,24,29,30], 57
Gate 5	[2,8,13,17,21,26], 91	[4,9,14,19,24,29], 108	[2,8,9,10,13], 95

Note. Second value represents the arrival delay per gate.

5 Conclusion

In this project, we applied Q-Learning and double DQN for AGAP. Our Q-Learning method was inappropriate for AGAP. Although categorization function can reduce the state space, it creates loss of information and this hinders the Q-Learning from finding the optimal policy. Next, we applied double DQN for basic MDP and feature-added MDP. In basic MDP, the policy derived from double DQN turned out be the greedy algorithm. Reasons for this behavior are that state only contains information about current stage and reward is given every stage. In feature-added MDP, there were features that contain information about future environment but the results were worse than basic MDP. For feature-added MDP to have better performance over greedy algorithm, reward function should be redefined as sparse reward function which evaluates the actions of one epoch at the final stage. Also different additional state features and network structure should be considered. [7] presented recurrent neural network for traveling salesman problem. Since AGAP is combinatorial optimization problem, this approach could be one possible option.

For future research, we are planning to solve stochastic AGAP by DQN. Stochastic AGAP is AGAP of which arrival time are uncertain parameters. It is expected that double DQN approach in AGAP can be easily generalized to stochastic AGAP by training networks from many AGAP scenarios. Also, investigating the interpretability of policy seems to be one interesting subject. By dissecting the policy resulted from deep learning, we can find unexpected behavior which could help us better understand the problem.

6 Reference

1. Daş, Gülesin Sena, Fatma Gzara, and Thomas Stützle. "A review on airport gate assignment problems: Single versus multi objective approaches." Omega 92 (2020): 102146.

2. Li, Mingjie, Jin-Kao Hao, and Qinghua Wu. "Learning-driven feasible and infeasible tabu search for airport gate assignment." European Journal of Operational Research (2021).

3. Jiaming, Zhao, et al. "Airport gate assignment problem with deep reinforcement learning." High Technology Letters 26.1 (2020): 102-107.

4. Li, Yijiang, John-Paul Clarke, and Santanu S. Dey. "Using submodularity within column generation to solve the flight-to-gate assignment problem." Transportation Research Part C: Emerging Technologies 129 (2021): 103217.

5. Mazyavkina, Nina, et al. "Reinforcement learning for combinatorial optimization: A survey." Computers & Operations Research 134 (2021): 105400.

6. Tassel, Pierre, Martin Gebser, and Konstantin Schekotihin. "A reinforcement learning environment for job-shop scheduling." arXiv preprint arXiv:2104.03760 (2021).

7. Bello, Irwan, et al. "Neural combinatorial optimization with reinforcement learning." arXiv preprint arXiv:1611.09940 (2016).